A Geometric Rendezvous-Based Domain Model for Data Transfer

Stuart R. Slattery
Department of Engineering Physics
University of Wisconsin - Madison
1500 Engineering Dr.
Madison, WI 53716
sslattery@wisc.edu

March 20, 2013

Abstract

The Data Transfer Kit (DTK) is a software component designed to provide parallel services for mesh and geometry searching and data transfer for arbitrary physics components. In many physics applications, the concept of mesh and geometry is used to subdivide the physical domain into a discrete representation to facilitate the solution of the model problems that describe it. Additionally, the concept of the field is used to apply degrees of freedom to the mesh or geometry as a means of function discretization. With the increased development efforts in multiphysics simulation, adaptive mesh simulations, and other multiple mesh/geometry problems, generating parallel topology maps for transferring fields and other data between meshes is a common operation. This document describes a domain model for mesh, geometry, fields, and parallel topology maps based on the concept of geometric rendezvous as implemented in DTK.

Acknowledgements

This work was funded by the Consortium for the Advanced Simulation of Light Water Reactors (CASL), a Department of Energy Innovation Hub [1].

Thanks to Paul Wilson (University of Wisconsin), Greg Davidson and Tom Evans (Oak Ridge National Laboratory), Roger Pawlowski (Sandia National Laboratories) and the CASL team for their help and guidance during the development of this work.

Contents

List of Figures

1 Introduction

In many physics applications, it is often desired to transfer fields (i.e. degrees of freedom or other data) between meshes or geometries that may or may not conform in physical space. In addition, for massively parallel simulations, it is typical that geometric domains not only do not conform spatially, but also that their parallel decompositions do not correlate and are independent of one another due to physics-based partitioning and discretization requirements. As an example, this situation can occur in multiphysics simulations where physics fields provide feedback between solution iterations or adaptive mesh simulations where fields must be moved between meshes after refining and coarsening. It is therefore desirable to have a set of tools to relate mesh and geometry of arbitrary parallel decomposition such that fields and other data can be transferred between them.

The Data Transfer Kit (DTK) is a software component designed to provide parallel services for mesh and geometry searching and data transfer based on the concept of the rendezvous decomposition [2]. To achieve a component design for use with arbitrary physics codes, general concepts of mesh, geometry, and fields are employed to provide access to these services. This document will outline the concepts of parallel communicators, mesh, geometry, fields, parallel topology maps, and the rendezvous decomposition and how they are modeled within the design of DTK. An example of data transfer using these concepts is also provided.

1.1 Basic Definitions

The following definitions are used throughout this document and serve as a basis for the domain model and discussion.

- **Communicator:** An object that allows communication of data between and controls the execution of operations on parallel processes.
- **Local Operation:** An operation that occurs within the context of a single process. All data and operations on that data are performed within the memory space of that process, independent of all other processes.
- **Global Operation:** An operation that occurs within the context of the entire parallel domain of the simulation. All data is potentially shared via communicator operations.
- **Ordinal:** A value that uniquely identifies an object from other objects of the same type. This number is a positive and real integer such that for a given ordinal, i, $i \in \mathbb{N}_0$.
- **Geometry:** An object or collection of objects that has n physical dimensions and a spatial domain $\Omega \in \mathbb{R}^n$ that is bounded by a boundary $\Gamma \in \mathbb{R}^n$.
- **Measure:** The measure of a geometry is defined as length in 1 dimension, area in 2 dimensions, and volume in 3 dimensions.

- **Vertex:** A zero-dimensional geometric object that has a globally unique ordinal and Cartesian coordinates that describe its geometric position. Vertices can have coordinates in 1, 2, or 3 dimensions.
- **Node:** A node is zero-dimensional mathematical object that describes the support points of a discretized function (e.g. the Lagrange basis points of a finite element discretization).
- **Mesh Element:** A mesh element, ω, is a discrete component of the spatial domain Ω that is constructed by vertices. These vertices have a canonical ordering and dimensionality that uniquely specifies the element type.
- **Element Topology:** A specific number of vertices with a specific canonical vertex ordering with a specific dimensionality. Examples include 4-vertex tetrahedrons and 3-vertex triangles. Every mesh element has an element topology.
- **Mesh:** A discrete representation of the n-dimensional spatial domain $\Omega \in \mathbb{R}^n$ consisting of an arbitrary number of mesh elements, ω. The mesh elements that construct a single mesh *may not intersect*. However, mesh elements may intersect if they belong to different meshes. Mesh can be considered a subset of geometry.
- **Mesh Block:** A collection of mesh elements that exist in the same mesh and have the same element topology. A mesh is composed of one or more mesh blocks.
- **Degrees of Freedom:** Discrete values generated by a physics computation that describe the discretization of a quantity over phase space.
- **Field:** A discrete representaion of a D-dimensional function, F, over the domain $\Omega \in \mathbb{R}^n$ such that $F(r) : \mathbb{R}^n \rightarrow \mathbb{R}^D, \forall r \in \Omega$. A field spans a domain and the discretization of that domain (e.g. a domain is resolved by mesh elements and therefore the field spans the mesh and its elements).
- **Evaluator:** An object that evaluates a field at a physical location, $\hat{r}$, in the spatial domain Ω by computing $F(\hat{r})$. This operation is valid $\forall \hat{r} \in \Omega$.
- **Source:** A geometry that owns a spatial domain, $\Omega_S \in \mathbb{R}^n$, over which an evaluator can be applied for a given field $F(s)$ where $s \in \Omega_s$.
- **Data Space:** A field, G, of dimension D, to which data can be applied such that $G(r) : \mathbb{R}^n \rightarrow \mathbb{R}^D, r \in \mathbb{R}^n$.
- **Target:** A geometry that owns a spatial domain, $\Omega_T \in \mathbb{R}^n$, over which a data space, $G(t)$ can be defined where $t \in \Omega_T$.
- **Geometric Rendezvous:** A geometric-based parallel redistribution of the original source and target geometries defined over the region $\Omega_R = \Omega_S \cap \Omega_T$.

- **Parallel Topology Map:** An operator, M, that defines the translation of a field, $F(s)$, from a source spatial domain, Ω_S, to a field, $G(t)$, in the target spatial domain Ω_T, such that $G(t) \leftarrow M(F(s))$ and $M : \mathbb{R}^D \rightarrow \mathbb{R}^D, \forall r \in \Omega_R$, where Ω_R is the geometric rendezvous of the source and target.

2 Communicators

A communicator is a concept that encapsulates ownership of operations in a parallel computation. A parallel communicator in the context of DTK can be taken as a direct reference to a Message Passing Interface (MPI) communicator [3]. A single communicator has a set of process space over which it may perform operations. Multiple communicators may exist in a single process space and own the same processes. For objects decomposed in parallel, the term global will be used to refer to concepts that apply across the entire parallel domain owned by a communicator while the term local will be used to refer to concepts that only exist in the domain of a single parallel process. A communicator is not required to encompass all of global process space. Based on this, we can consider union and intersection operations. Two communicators will be deemed equivalent if they own the same set of processes.

2.1 Communicator Union

A union operation will compute the union in process space of all communicators involved. This will be a common operation for cases where one particular geometric component in the data transfer operation is decomposed over a different communicator than another geometric component. Figure 1 provides an example of a union operation. In this example, two communicators, A and B, are defined within the domain of a global communicator (typically, but not necessarily MPI_COMM_WORLD) but do not encapsulate all of it. All processes that exist either in communicator A or B will be used to form the new union communicator (C_{union}) as a subset of the global communicator (C_{global}) such that $C_{union} = A \cup B$ and $C_{union} \in C_{global}$. This operation is only valid if $A \in C_{global}$ and $B \in C_{global}$.

2.2 Communicator Intersection

An intersection operation will compute the intersection in process space of all communicators involved. Figure 2 provides an example of an intersection operation. In this example, two communicators, A and B, are defined within a global communicator but do not encapsulate all of it. All processes that exist in both communicator A and B will be used to form the intersection communicator. The intersection communicator ($C_{intersect}$) is a subset of the global communicator (C_{global}) such that $C_{intersect} = A \cap B$ and $C_{intersect} \in C_{global}$. This operation is only valid if $A \in C_{global}$ and $B \in C_{global}$.

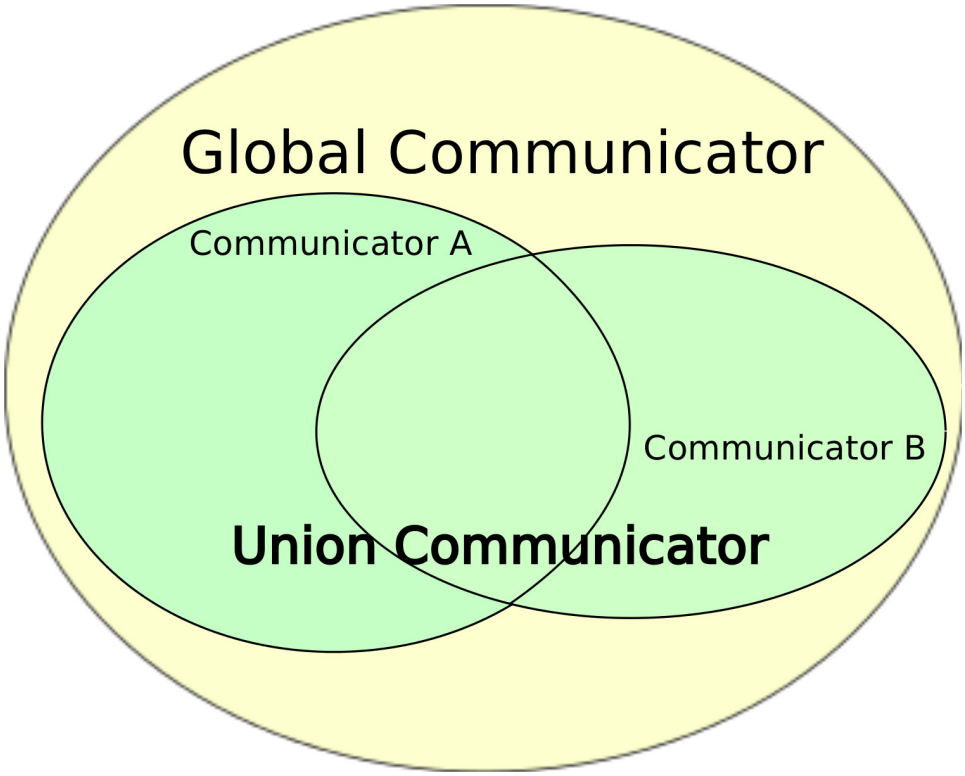

Figure 1: *Communicator union operation diagram.*
In this example, communicator A and communicator B are contained within the global communicator but do not encapsulate all of it. Their combined communication spaces form the union. The union is a subset of the global communicator.

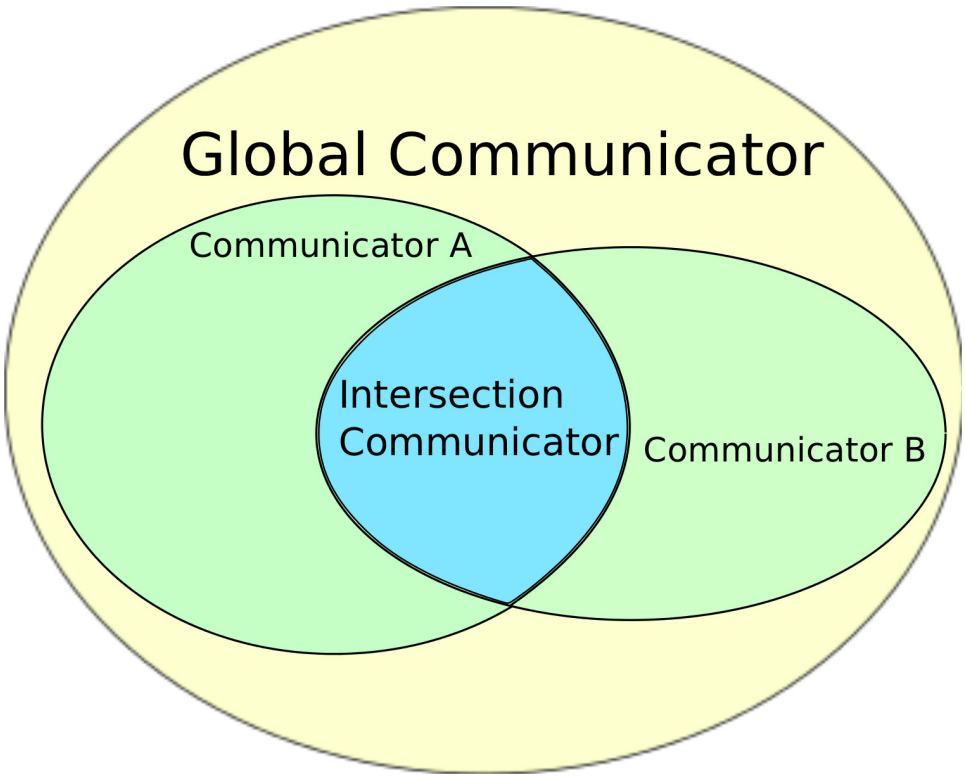

Figure 2: *Communicator intersection operation diagram.*
In this example, communicator A and communicator B are contained within the global communicator but do not encapsulate all of it. The blue communication space forms the intersection.

3 Mesh

In order to access DTK mesh services, a subset of the information needed to describe the mesh is required. This subset consists of vertices and their coordinates, elements and the vertices that construct them, and the communicator over which they are defined. The vertices that construct an element have both a canonical ordering consistent across all elements of that topology in a mesh and a permutation list that describes how this ordering varies from DTK canonical ordering. This subset has been demonstrated as sufficient for applying data transfer algorithms [4] and can be formulated in such a way that algorithms can be generated that are data structure neutral [5]. In addition, for meshes that contain multiple element topologies, the concept of mesh blocked by element topology is utilized.

3.1 Mesh Vertices

Vertices are the lowest level geometric component of the mesh. All vertices have a globally unique ordinal serving as an identification number for the vertex in global operations. A vertex can have 1, 2, or 3 dimensions but all vertices in a mesh must have the same dimension. To specify its geometric position, each vertex has Cartesian (x,y,z) coordinates. A vertex must provide only the coordinates for the specified vertex dimension, no more or no less (e.g. a 2 dimensional vertex must provide x and y coordinates but not a z coordinate). A vertex may be repeated any number of times across the parallel domain with unlimited local and global instances. However, every vertex with the same globally unique ordinal must have the same coordinates. We make a distinction here between vertices and nodes. In the context of DTK, a vertex is purely a geometric object. It describes the spatial positioning and geometric bounds of an element. A node is purely a mathematical object. It descrbibes the descretization associated with a particular element described within the natural coordinate system of that element. It is possible that in the physical coordinate frame that a node and vertex may occupy the same geometric location, however DTK does not consider nodes in its formulation.

As an example, consider Figure 3 depicting a series of vertices contained in a mesh. Each vertex provides a globally unique ordinal and set of 3 dimensional coordinates.

3.2 Mesh Elements

Elements are the second level of abstraction in the mesh description above vertices. All elements have a globally unique ordinal serving as an identification number for the element in global operations. This globally unique ordinal can be the same as a globally unique ordinal for a vertex in the mesh as DTK distinguishes between vertices and elements. An element has a topology defining its physical structure (e.g. tetrahedron, hexahedron, etc.) and a number of vertices needed to generate that topology. Elements are constructed from vertices via a connectivity list. The connectivity list for a particular element will contain the unique vertex global ordinals that construct its linear form. An element may be repeated any number of times

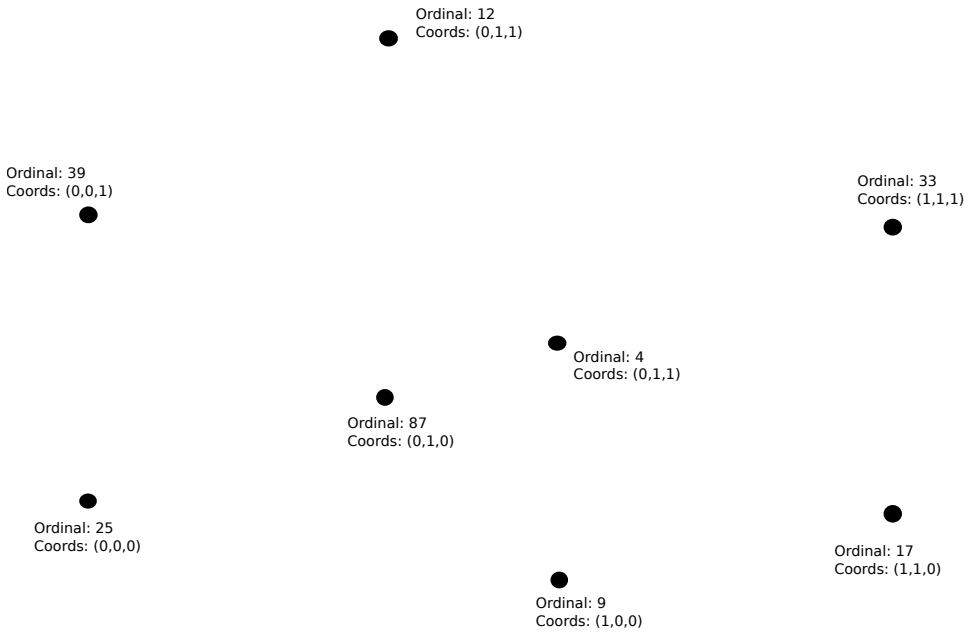

Figure 3: *Basic vertex description for a mesh.*
Each vertex is required to have a unique global ordinal, a specified dimensionality, and Cartesian coordinates corresponding to that dimensonality. The vertices in this example are 3 dimensional.

across the parallel domain (i.e. it may have unlimited local instances), however, every globally unique ordinal must have the same connectivity list associated with it. For consistency, DTK uses the MoaB Canonical Numbering (MBCN) scheme as a canonical ordering scheme [6]. Each element in a client mesh can be described with a connectivity list using any canonical scheme of choice, however, the relationship between this canonical numbering scheme and the DTK canonical numbering scheme must be made available. Each element topology is therefore also described by a permutation list. A permutation list specifies the variation in ordering between the DTK canonical numbering scheme and the client canonical numbering scheme. A permutation list must be described globally, regardless of whether or not elements exist on a particular process. See Appendix A for canonical element topologies as defined by DTK. These are the canonical vertex orderings that must be used when generating a permutation list for a client element topology. Mesh elements may not intersect any other elements in a single mesh description. An element may intersect other elements if those elements exist in another mesh (this is in fact a common situation in data transfer).

Consider the continuation of our example in Figure 4 showing a linear hexahedron element generated from the vertices in Figure 3. The element has been given a unique global ordinal and the connectivity and permutation lists have been specified. The connectivity list specifies an element construction from counter-clockwise movement around the bottom face and then counter-clockwise movement around the top face that is native to the client. The MBCN canonical ordering for linear hexahedrons

is given at the vertices. Note that MBCN ordering instead uses clockwise rotation around the bottom and top faces to construct the element. This difference in ordering is specified by the given permutation list.

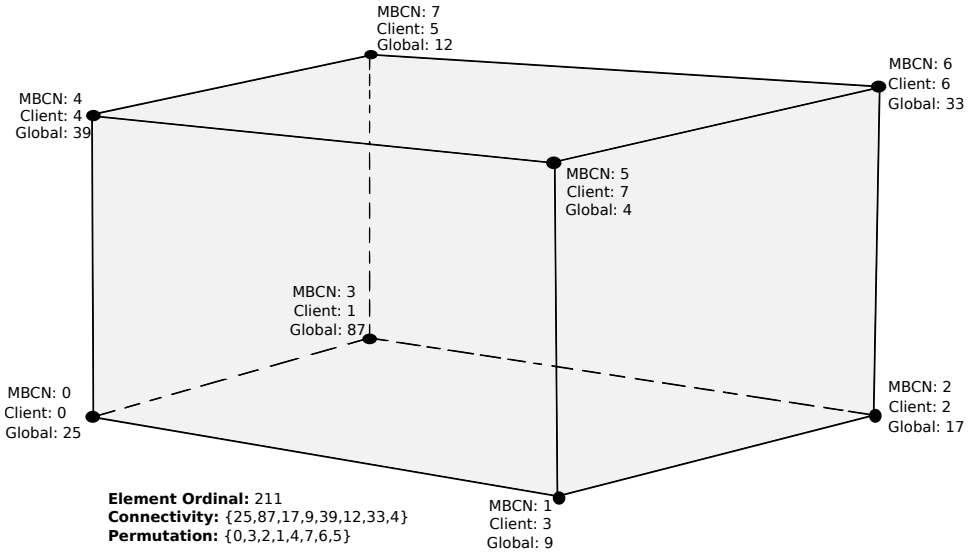

Figure 4: *Basic element description for a mesh.*
Each element is required to have a unique global ordinal, and a specified connectivity and permutation list. The MBCN canonical vertex ordinals used by DTK, the client canonical vertex ordinals, and the client global vertex ordinals are specified at the vertices.

3.3 Mesh Blocks

At the highest level of abstraction, the mesh is composed of mesh blocks. All elements in a block must have the same topology and number of vertices. A mesh may contain as many blocks as desired. Multiple blocks with the same mesh topology may exist. Vertices may be repeated in different mesh blocks provided that they maintain the same unique global ordinal and coordinates. Elements may be repeated in different mesh blocks provided that they maintain the same unique global ordinal and connectivity list. All elements and vertices in all blocks of the mesh must have the same dimension. Multiple mesh blocks may exist in the same spatial region as they are merely a means of subdividing the mesh into groups of elements based on their topology. This behavior will be the common when hybrid meshes are involved (e.g. a mesh that contains hexahedrons and tetrahedrons). Mesh blocks may be either structured or unstructured.

As an example, consider the 2 dimensional hybrid mesh presented in Figure 5. This mesh contains both quadrilateral (blue) and triangle (red) elements that share connecting vertices. In this case, all quadrilaterals should be specified in a single mesh block and all triangles specified in another mesh block. The vertices shared by these two mesh blocks may be repeated in either block.

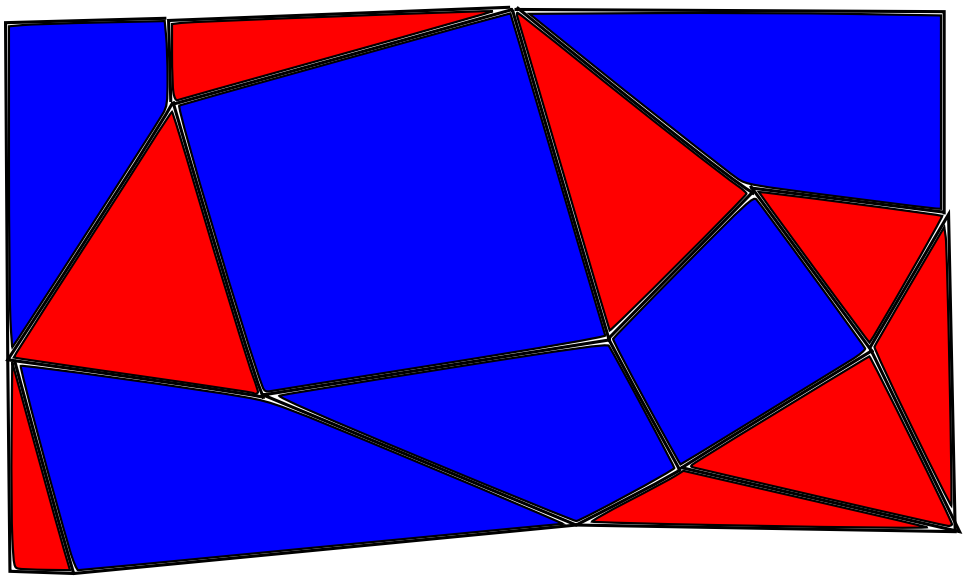

Figure 5: *Hybrid mesh example.*
Quadrilaterals (blue) must be specified in a different mesh block than the triangles (red). Both blocks can contain the mutual mesh vertices that construct their elements.

3.4 Parallel Decomposition

Mesh blocks and the elements and vertices they contain may be partitioned in any fashion provided that all vertices, elements, and blocks of a mesh description exist in a communication space operated on by the same parallel communicator. Different blocks in a single mesh description may not have different communicators. Each mesh description may have its own communicator. Global knowledge of the parallel decomposition of a given mesh description is not required. Only local mesh data access along with the proper communicator is required.

4 Geometry

A geometry is simply an object or collection of objects that has n physical dimensions and a spatial domain $\Omega \in \mathbb{R}^n$ that is bounded by a boundary $\Gamma \in \mathbb{R}^n$. Concrete examples of geometries in 3 dimensions include cubes and cylinders. A geometry can have 1, 2, or three dimensions. All geometry objects have a globally unique ordinal serving as an identification number for the geometry in global operations. To specify the general position in space of the geometry, each object is required to have a centroid given in Cartesian coordinates with (x) given for 1 dimensional geometries, (x,y) for two dimensions, and (x,y,z) for 3 dimensions. A measure is also specified for each geometry where the measure is defined as length in 1 dimension, area in 2 dimensions, and volume in 3 dimensions. In addition to this data, a geometry must be able to provide a Cartesian axis-aligned bounding box that encapsulates the entire geometry. For geometric search operations to be performed, a geometry must be able to determine if a given point of the same dimensionality as the geometry is contained within the boundary of the geometry (i.e. $\hat{r} \in \Omega$).

5 Fields

In the most general sense, a field refers to the degrees of freedom computed by a physics code or the responses derived from those degrees of freedom that have been discretized across the domain [7]. The field is implicitly bound to the geometric domain through the degrees of freedom and their association with a mesh or other geometric components. In a physics simulation, examples of degrees of freedom include pressure and velocity distributions and examples of computed reponses include heat flux or reaction rates. In order to access DTK field services, a subset of information needed to describe the field is required. A field has a dimension of arbitrary size. As examples, for scalar fields this dimension is 1, for 3-vectors (such as the velocity example above in a 3 dimensional computation) the dimension is 3, and for a 3×3 tensor the dimension is 9. All local instances of the field must have the same dimension. A field can have an arbitrary number of local degrees of freedom and this size can differ from local domain to local domain. No knowledge of the global field decomposition is required, however, it must exist on a single communicator.

5.1 Field Evaluations

The actual discretization of the field is not explicitly formulated. Rather, access to discretization of fields and the associated data is generated through function evaluations at points in physical space. Consider a D-dimensional function $F(r)$ of arbitrary discretization over the spatial domain $\Omega \in \mathbb{R}^n$ where $r \in \mathbb{R}^n$ and $F : \mathbb{R}^n \rightarrow \mathbb{R}^D$. Via polynomial interpolation, projection, or any other means necessary to most appropriately reflect the discretization of $F(r)$, it then follows that evaluation operations of the following type can be performed:

$$\hat{f} \leftarrow F(\hat{r}), \forall \hat{r} \in \Omega \tag{1}$$

where $\hat{r} \in \mathbb{R}^n$ is a single point and $\hat{f} \in \mathbb{R}^D$ is representative of the function $F(r)$ evaluated at $\hat{r}$. This operation is not valid for $\hat{r} \notin \Omega$. In the context of Ω discretized by a mesh, these evaluations can instead be written in terms of a single mesh element, $\omega \in \Omega$:

$$\hat{f} \leftarrow F(\hat{r}), \forall \hat{r} \in \omega \tag{2}$$

This operation is then not valid for $\hat{r} \notin \omega$. If $\hat{r} \notin \omega$ and $\hat{r} \notin \Omega$, then alternative schemes may be chosen, such as extrapolation, in order to apply the field to $\hat{r}$. An evaluator is an object that drives the application of Eq (2).

5.2 Field Integrations

As a complement to field evaluations at points, fields may also be integrated over a region of space, Ω:

$$f_\Omega = \int_\Omega F(r) dr \,, \tag{3}$$

where f_Ω is now representative of the integral of the field $F(r)$ over the domain Ω. In the context of Ω discretized by a mesh, these integrals can instead be written in terms of a single mesh element, $\omega \in \Omega$:

$$f_\omega = \int_\omega F(r) dr \,, \tag{4}$$

where the integral over Ω will be the measure-weighted summation of all element integrals:

$$f_\Omega = \frac{1}{m_\Omega} \sum_i f_{\omega_i}, \ \forall \omega_i \in \Omega \,, \tag{5}$$

where m_Ω is the measure of the geometric domain.

5.3 The Relationship Between Mesh and Fields

The relationship between the mesh and the field in DTK is implicitly defined. As stated in Eq (1), a field evaluation is only valid over a particular spatial domain Ω. We define the mesh over that same domain Ω. In addition, as stated in Eq (2), the discrete element components of the mesh, ω, also have a relationship with the field. Given a particular element in the mesh such that $\omega \in \Omega$, the field is then also bound to those discrete components. The discrete form of the field then forms an aggregate with the discrete form of the mesh. The responses derived from that field also are also bound to the mesh through this relationship.

6 Geometric Rendezvous

Relating two non-conformal meshes will ultimately require some type of evaluation algorithm to apply the data from one geometry to another as specified by Eq. (1). To drive these evaluation algorithms, the target objects to which this data will be applied must be located within the the source geometry. In a serial formulation, efficient search structures that offer logarithmic asymptotic time complexity are available to perform this operation. However, in a parallel formulation, if these two geometries are arbitrarily decomposed, geometric alignment is not likely and a certain degree of communication will be required. A geometric rendezvous manipulates the source and target geometries such that all geometric operations have a local formulation.

A geometry that is associated with the providing data through function evaluations will be referred to as the source geometry while the geometry that will be receiving the data will be referred to as the target geometry. Although explicitly formulated with a source mesh and target vertices below, these concepts can be applied to geometric structures beyond mesh and vertices.

6.1 The Rendezvous Algorithm

The geometric rendezvous concept uses a global formulation for the data transfer while maintaining a local formulation for the geometric search operations. In DTK, the following algorithm developed by Plimpton et. al. [2] generates the rendezvous decomposition through global operations in order to achieve a local framework for geometric operations.

1. Compute a box that bounds the source and target geometry intersection.
2. Create a rendezvous decomposition by performing recursive coordinate bisectioning on the source geometry.
3. Send source geometry from the source decomposition to rendezvous decomposition.
4. Clone source geometry components which overlap into nearby recursive coordinate bisectioning sub-domains.
5. Build a kD-tree with the local mesh in each rendezvous partition.

This algorithm is elaborated in more detail in the following paragraphs.

Step 1: Bounding box construction. To begin, a global bounding box for the source and target geometries is constructed using their nodal data. These two bounding boxes are then intersected to produce the bounding box around the intersection of the two geometries. Those source or target vertices that are not inside this intersection bounding box are not considered for the remainder of the algorithm. This step is motivated by the fact that for many classes of data transfer problems, such as

2 dimensional surface transfer in a full 3 dimensional problem, only a small subset of the source and target geometries will be used. This will reduce the number of search operations and communication operations.

Step 2: Rendezvous decomposition generation. Recursive coordinate bisectioning (RCB) is used the create a new decomposition with the source mesh vertices [8]. We choose RCB here to repartition the source mesh to achieve a more load-balanced decomposition for geometric operations. For many physics codes, the parallel decomposition of the mesh is generated with physics-related efficiencies in mind. For the purposes of geometric operations related to search mesh data structures, the geometric nature of RCB alleviates the potential inefficiencies for geometric operations that may be incurred by using the original source decomposition. This new decomposition, the rendezvous decomposition, will serve as an intermediary between the original source and target decompositions.

Step 3: Send source elements to rendezvous decomposition. The RCB decomposition is generated only from source mesh vertex information. The element information must be sent to the rendezvous decomposition and reconstructed for the point location process.

Step 4: Clone source elements that overlap RCB sub-domains. Because RCB was performed using source nodal data, there will be source elements that span the boundary between two or more RCB sub-domains. When this occurs, the source elements will be repeated in each RCB sub-domain in which their connectivity vertices exist.

Step 5: Build a kD-tree with the local mesh in each rendezvous partition. On each rendezvous process, the local mesh is searched with the local target vertices. A kD-tree is generated using the local mesh for a fast proximity search of the domain that computes a small subset of the local mesh that resides near the vertex [9].

6.2 The Rendezvous Decomposition

Using the above algorithm, a secondary decomposition of a subset of the source mesh is generated forming the rendezvous decomposition. The rendezvous decomposition is encapsulated as a separate entity from the original geometric description of the domain. It can be viewed as a copy of the source mesh subset that intersects the target geometry. This copy has been repartitioned in a way that fundamental geometric search operations are local and proceed globally in a load balanced fashion.

The rendezvous decomposition has several properties. It is defined over a communicator that encapsulates the union of the communication spaces owned by the source and target geometries. It is defined inside of a global, axis-aligned bounding box that bounds the intersection of the source and target geometries. The decomposition is of the same dimension as the source and target geometries. A rendezvous decomposition

cannot be generated with source and target geometries of different dimensions (e.g. a 3 dimensional source geometry and a 2 dimensional target geometry cannot be used to generate a rendezvous decomposition).

6.3 Searching the Rendezvous Decomposition

By implementing the above algorithm, we effectively have a search structure that spans both parallel and physical space. We first search parallel space by querying the rendezvous decomposition generated during repartitioning. Global recursive coordinate bisectioning parameters are maintained for global partitioning information, meaning that a desintation process in the rendezvous decomposition can be determined for any point on any process. Although this is a search over parallel space, because of the geometric nature of the rendezvous decomposition it is also a search over physical space with each process in the rendezvous decomposition owning a specific subset of the mesh (with marginal overlap at the boundaries).

Once points have been accumulated in the rendezvous decomposition, the local kD-tree that is formed over the local mesh can be utilized. By searching the kD-tree in logarithmic time, a subset of the mesh that is in the vicinity of the target point is generated. This subset, which is typically much smaller than the mesh owned by a particular rendezvous procees, is then searched with a more expensive point-in-element operation that transforms the vertex into the reference frame of each element in the subset with a Newton iteration strategy. This mapped point is then checked against the canonical reference cell of that element's topology to determine if the vertex is contained within.

7 Parallel Topology Maps

A parallel topology map is an operator, M, that defines the translation of a field, $F(s)$, from a source spatial domain, Ω_S, to a field, $G(t)$, in the target spatial domain Ω_T, such that $M : \mathbb{R}^D \rightarrow \mathbb{R}^D, \forall r \in [\Omega_S \cap \Omega_T]$, using both geometric and parallel operations. There are 3 types of physics-based parallel topology maps: shared domain maps, interface maps, and network maps [7]. Currently, DTK only supports shared domain maps.

7.1 Shared Domain Problems

A shared domain problem is one in which the geometric domains of the source and target intersect over all dimensions of the problem. Figure 6 gives an example of a shared domain problem in 3 dimensions. Here, $\Omega(S)$ (*yellow*) is the source geometry, $\Omega(T)$ (*blue*) is the target geometry, and $\Omega(R)$ (*red*) is their intersection and the shared domain over which mapping and the rendezvous decomposition will be generated.

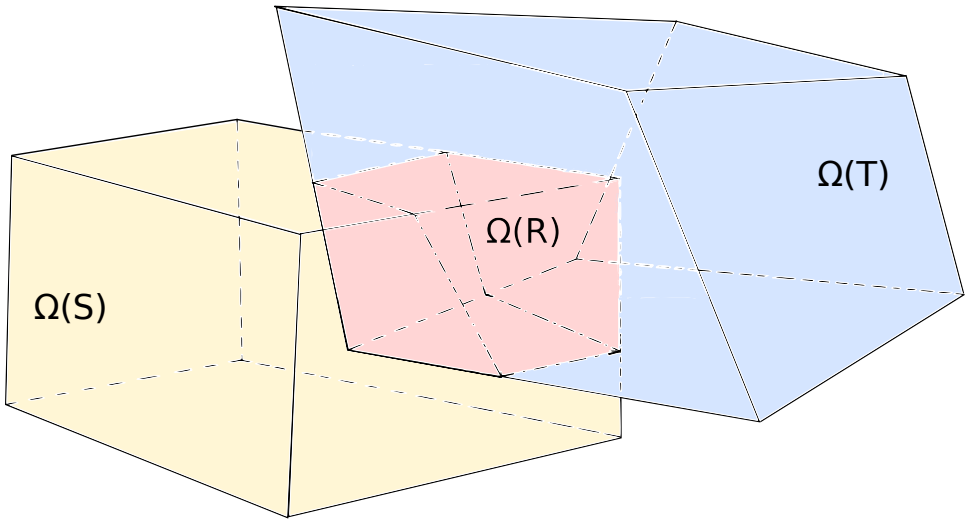

Figure 6: *Shared domain example.*
$\Omega(S)$ (yellow) is the source geometry, $\Omega(T)$ (blue) is the target geometry, and $\Omega(R)$ (red) is the shared domain.

The shared domain map has several properties. It is defined over a communicator that encapsulates the union of the communication spaces owned by the source and target geometries. The map is of the same dimension as the source and target geometries. A shared domain map cannot be generated with source and target geometries of different dimensions (e.g. a 3 dimensional source geometry and a 2 dimensional target geometry cannot be used to generate a shared domain map). The source and target domains may have an arbitrary parallel decomposition unrelated to one another. There are 3 types of shared domain mapping algorithms implemented in DTK: the traditional mesh-based rendezvous algorithm, the geometry source map, and the integral assembly map.

7.1.1 Mesh-Based Shared Domain Mapping Algorithm

The following mapping algorithm is applied using the rendezvous decomposition to build a parallel topology map for shared domain problems where both the source and target geometries are represented by mesh.

1. Determine which target geometry components to operate on by finding those that reside in the rendezvous decomposition global bounding box.
2. Find which rendezvous sub-domain each target geometry component is inside of and determine the process which owns this sub-domain.
3. Send target geometry from target decomposition to rendezvous decomposition.
4. Find which target geometry component is inside which source geometry component.
5. Send source/target pairings from rendezvous decomposition to source decomposition.

Step 1: Get the target vertices that are in the rendezvous decomposition bounding box. We will only operate on those vertices that are contained within the rendezvous decomposition bounding box. Those that are not will not participate in mapping.

Step 2: Find rendezvous subdomain for target vertices. On each target process, the RCB decomposition is searched to find the destination process for the target vertex in the rendezvous decomposition.

Step 3: Send target vertices to rendezvous decomposition. Each target vertex is moved to the appropriate RCB sub-domain for the point location step.

Step 4: Find the source elements in which target vertices reside. We now have source element information and target vertex information in the rendezvous decomposition. On each rendezvous process, the local mesh is searched with the local target vertices using the kD-tree and underlying point-in-element functionality.

Step 5: Send element/vertex pairs back to original decompositions. The source element/target vertex pairs are the map in this case and they will be used to drive the function evaluation routines. These pairings must be communicated from the rendezvous decomposition back to the source/target decompositions to complete the mapping.

7.1.2 Geometry Source Map

A slight variation of the mesh-based rendezvous algorithm is one in which the source field is defined over a set of geometric structures instead of a mesh. There are two potential use cases for this type of map: one in which the target is a set of points at which the source function should be evaluated and another in which the target is a group of geometric objects identical to those in the source but with a possibly different parallel distribution. For the first case, the general point location algorithm is the same as above but instead an array of general geometric structures is searched. Those geometries may span the domain of more than one parallel process, but if they do then their entire description is required on each process. The result of the mapping sequence is a set of geometry/target point pairs used to drive the evaluation algorithms for the source functions within the geometries.

For the second use case, a few assumptions about the source and target geometries are used to enable geometry-to-geometry mappings where the parallel decompositions of the geometries are not known a priori. The first assumption requires both the source and target geometries to occupy the same physical space (i.e. a cylinder in the source code has the same height, radius, and centroid as the cylinder in the target code). If this assumption is valid, it then follows that if the centroids of the target geometries are supplied as the target points for this map, then the source geometries in which they are located correlate to their identical counterparts in target geometries. In this context, the field evaluations are then interpreted as simply gathering the geometry-related quanities (which still may involve a function evaluation or integration) and redistributing them to the target decomposition.

7.1.3 Integral Assembly Map

In certain cases of data transfer, a target physics code may desire quanities averaged over the spatial domain to be applied to its geometry. If the source code uses the same geometric representation, then the geometry source map may be used as described above. However, if the source code uses a mesh-based discretization of the spatial domain, then this averaged quantity will require the following measure weighted integral to be performed over the source field defined on the mesh elements and appropriately assembled in parallel over the proper geometric structures:

$$f_\Omega = \frac{\int_\Omega F(r)dr}{\int_\Omega dr} \,, \tag{6}$$

which if interpeted as an integral of a field defined over a mesh gives:

$$f_\Omega = \frac{\sum_i \left[\int_{\omega_i} F(r)dr \right]}{m_\Omega} \,, \tag{7}$$

where m_Ω is the measure of the target geometry and the integrations are as outlined in § 5.2. If the mesh is assumed conformal to the geometry then:

$$m_\Omega = \sum_i m_{\omega_i}, \ \forall \omega_i \in \Omega \,, \tag{8}$$

where m_{ω_i} is the measure of the i^{th} mesh element that is physically within the geometry.

As the source mesh elements and target geometries may have an arbitrary parallel decomposition, the rendezvous decomposition is used to correlate the two. In this mapping algorithm, the rendezvous decomposition is used to determine which mesh elements are contained within each geometric entity. As it is assumed that the mesh is conformal to the geometry, the vertices of the mesh elements are checked for point inclusion with the geometry where either a single vertex or all vertices may be required for element inclusion. Once this parallel mapping information is acquired, the measures of the target geometries are approximated by gathering the measures of the individual elements that compute their sums. To drive the application of the map, we use a mesh-element integration function for each field in the source code to acquire $m_{\omega_i} \int_{\omega_i} F(r)dr$ in the source decomposition. These measure-weighted element integrals are then moved into the target decomposition and the sum in Eq (7) is computed for each target geometry using the computed geometry measures.

7.1.4 Handling Target Objects Outside of the Source Domain

The mesh-based shared domain map and the geometry source map both have several steps in which target objects may or may not be found in the source mesh. Both the RCB decomposition search and kD-tree search have the potential to return target objects that were not found in the source mesh. The source function will not be evaluated at these points as they are not in the domain Ω_S, and therefore the evaluation operation will not be valid. However, a list of these points in the target decomposition may be generated for further processing by the client.

8 An Example

Consider the following 2-dimensional shared domain problem that utilizes the above concepts and the mesh-based rendezvous algorithm to map a field from one mesh to another. Note that the data presented in this example was generated externally from DTK for the purpose of visualizing the algorithm.

A linear triangle mesh is chosen for the source mesh as shown in Figure 7. The source mesh is decomposed over 4 parallel processes with each process operating on elements of a single color. The target mesh is defined with linear quadrilaterals and is given by Figure 8. Here, the decomposition is also over 4 processes but with a different spatial decomposition. We therefore expect communication between all 4 processes in this example.

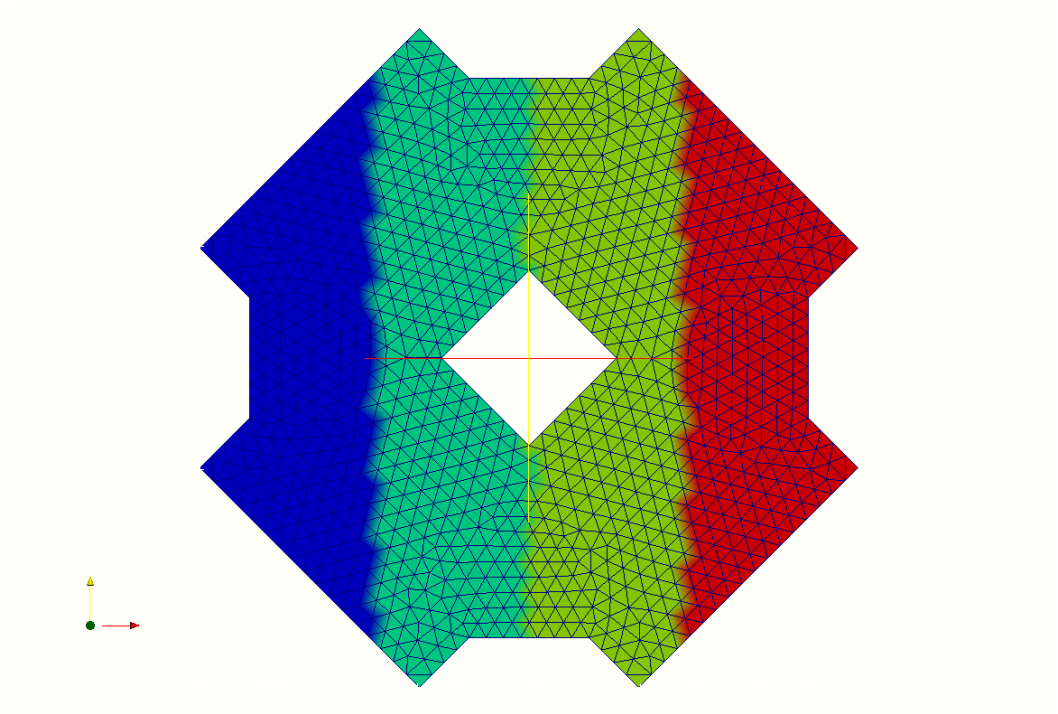

Figure 7: *Source mesh for 2D shared domain example.*
Each color represents the portion of the mesh owned by each parallel process.

Using the rendezvous algorithm, we repartition the source and target geometries such that they align in a new, geometric-based partitioning as shown in Figure 9. In parallel search operations, we then first search the partition domains to get the rendezvous process containing a particular target object. We then communicate this point to the proper process in Figure 9 and search that process' domain using the kD-tree. The source element/target object pairs generated from this search process are then communicated back to the process that owns the source in element in the decomposition given by Figure 7.

Shown in Figure 10, we define a field over the source mesh, in this case the peaks function, to transfer to the target mesh. This function is valid only over the domain

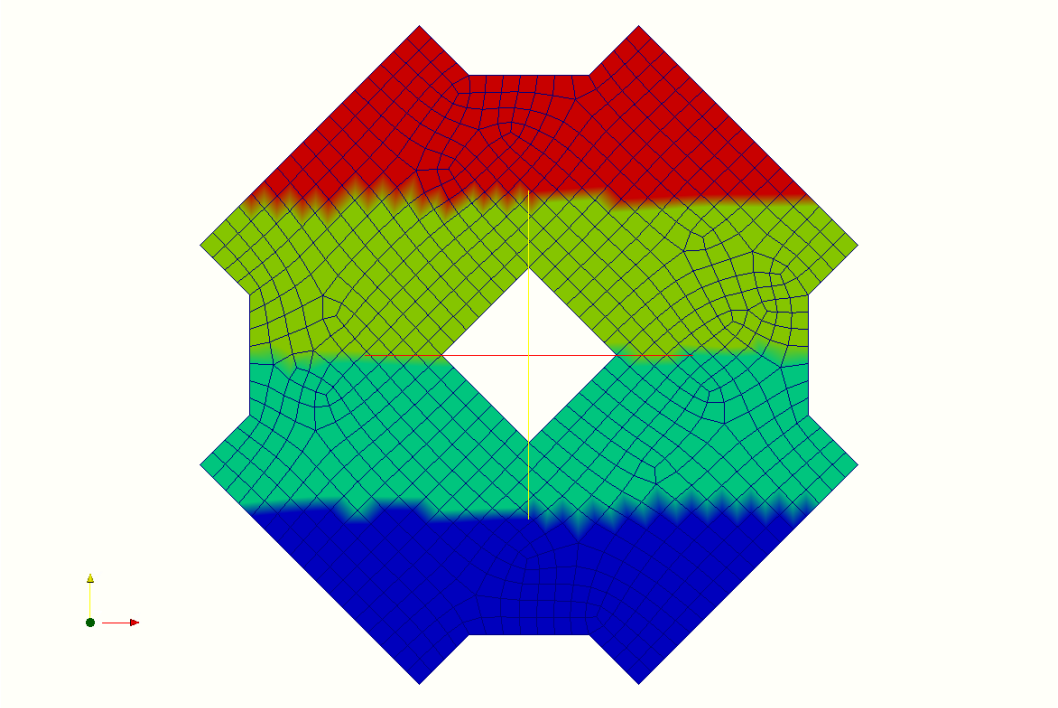

Figure 8: *Target mesh for 2D shared domain example.*
Each color represents the portion of the mesh owned by each parallel process.

of the mesh and can be evaluated at any 2-dimensional point that resides in a valid element.

Finally, an evaluator associated with the source mesh is used to drive peaks function evaluations on the target mesh vertices in the source decomposition given by Figure 7. These are then communicated back to the target decomposition, resulting in the field transferred as shown in Figure 11.

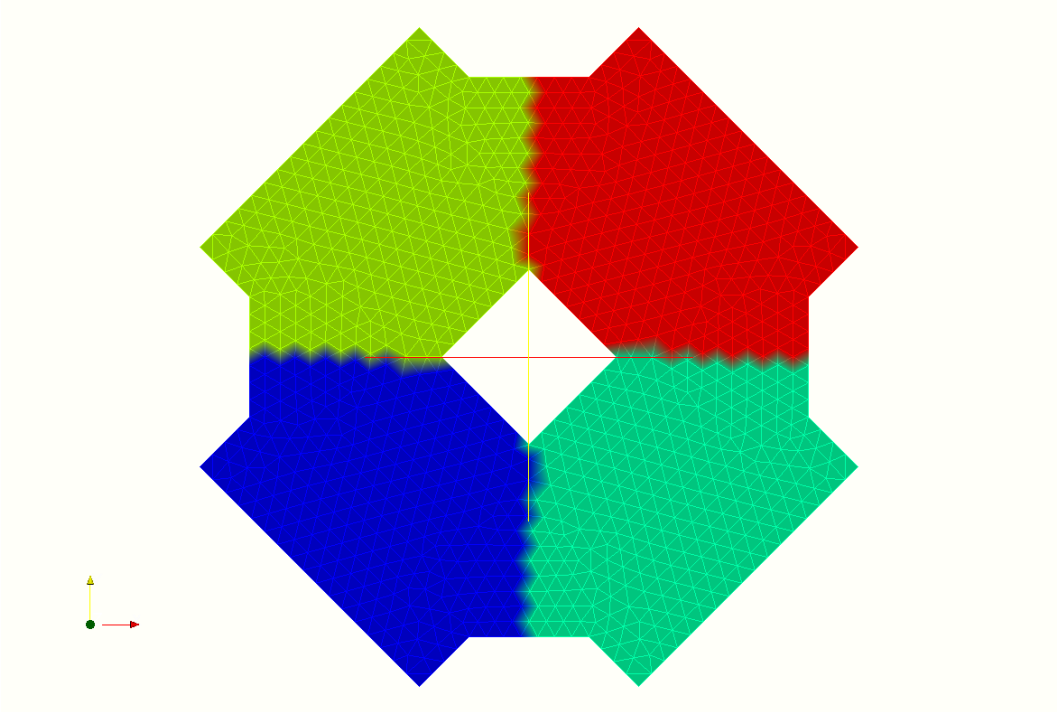

Figure 9: *Rendezvous decomposition for 2D shared domain example. Each color represents the portion of the mesh owned by each parallel process. Both the source and target geometries are repartitioned in this fashion such that all search operations are on-process.*

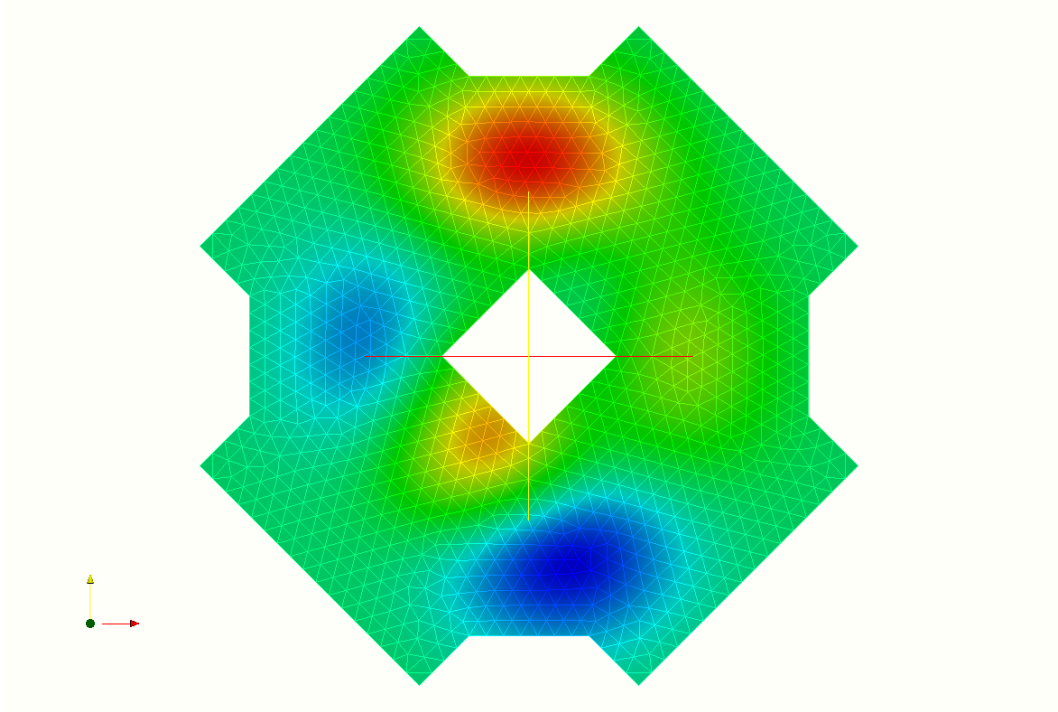

Figure 10: *Source function for 2D shared domain example. A 2D peaks function is defined over the source domain.*

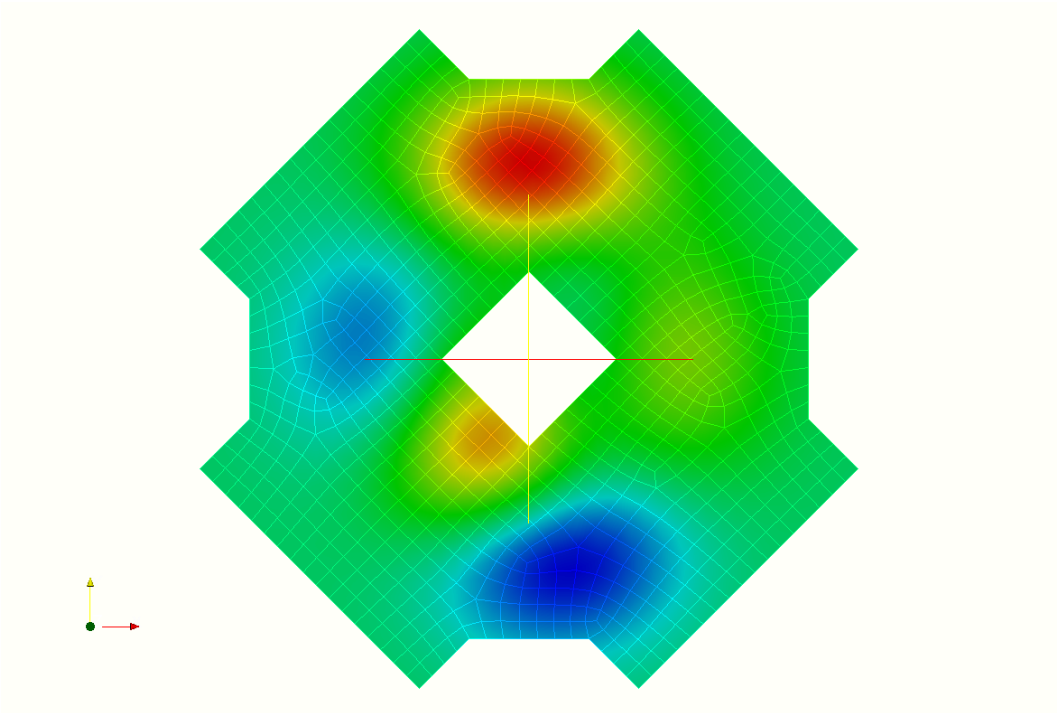

Figure 11: *Target function for 2D shared domain example. A 2D peaks function is transferred to the target domain.*

References

[1] "Consortium for the advanced simulation of light water reactors online: http://www.casl.gov," 2012.

[2] S. Plimpton, B. Hendrickson, and J. Stewart, "A parallel rendezvous algorithm for interpolation between multiple grids," *Journal of Parallel and Distributed Computing*, vol. 64, pp. 266–276, 2004.

[3] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: portable parallel programming with the message-passing interface*. MIT Press, 1994.

[4] J. Stewart and H. Edwards, "A framework approach for developing parallel adaptive multiphysics applications," *Finite Elements in Analysis and Design*, vol. 40, pp. 1599–1617, 2004.

[5] K. Chand, L. Diachin, C. Ollivier-Gooch, E. Seol, M. Shephard, T. Tautges, and H. Trease, "Toward interoperable mesh, geometry and field componenents for PDE simulation development," *Engineering with Computers*, vol. 24, pp. 165–182, 2008.

[6] T. Tautges, "Canonical numbering systems for finite-element codes," *International Journal for Numerical Methods in Biomedical Engineering*, vol. 20, pp. 1559–1572, 2009.

[7] R. Pawlowski, R. Bartlett, N. Belcourt, R. Hooper, and R. Schmidt, "A theory manual for multi-physics code coupling in lime," no. SAND2011-2195, 2011.

[8] M. Berger and S. Bokhari, "A partitioning strategy for nonuniform problems on multiprocessors," *IEEE Transactions on Computing*, pp. 570–580, 1987.

[9] J. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, pp. 509–517, 1975.

A DTK Element Topologies

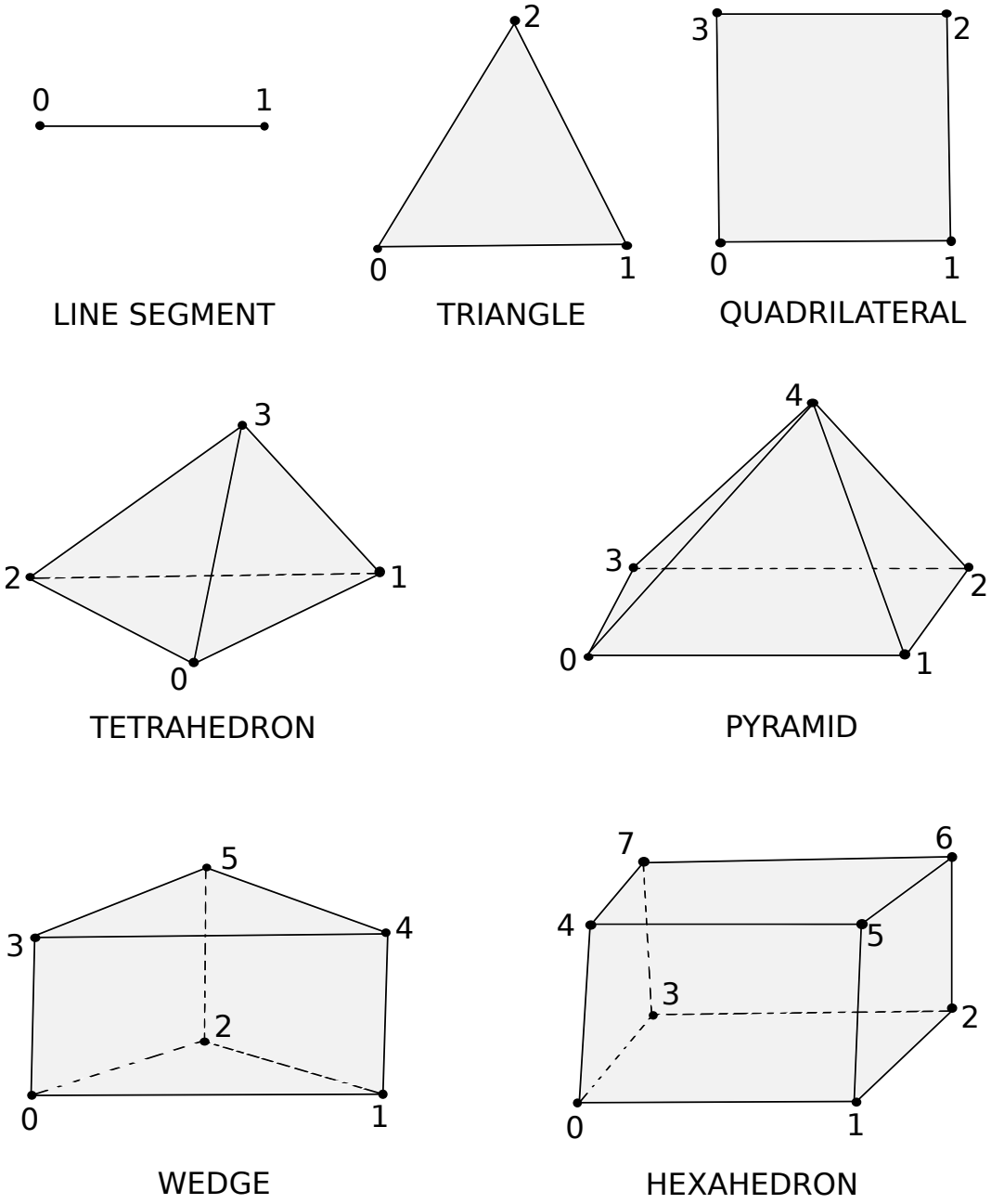

Figure 12: *Canonical vertex connectivity schemes for elements in DTK.*